# Marte CCSL to execute East-ADL Timing Requirements

Frédéric Mallet, Marie-Agnès Peraldi-Frati, Charles André

# Marte CCSL to execute East-ADL Timing Requirements

Frédéric Mallet, Marie-Agnès Peraldi-Frati, Charles André
Aoste Project I3S-INRIA, Université de Nice-Sophia Antipolis
INRIA Sophia Antipolis Méditerranée, FRANCE
{fmallet,maperal,candre}@sophia.inria.fr

## Abstract

*In the automotive domain, several loosely-coupled Architecture Description Languages (ADLs) compete to provide a set of abstract modeling and analysis services on top of the implementation code. In an effort to make all these languages, and more importantly their underlying models, interoperable, we use the UML Profile for MARTE as a pivot to define the semantics of these models.*

*In this paper, we particularly focus on East-ADL2. We discuss the benefits of having an integrated, MARTE-centered, approach. We give a formal semantics of East-ADL2 timing requirements. Relying on this semantics, several kinds of analysis are possible. Requirements become executable and simulations are run. A constraint solver is used to detect logical inconsistencies. Our proposal is illustrated on an Anti-lock Braking System (ABS).*

## 1. Introduction

Architecture Description Languages (ADLs) are more and more accepted as means to manage the engineering information related to automotive electronics and deal with the increasing complexity of the automotive software [7]. Several loosely-related ADLs are competing in that area (AADL [11], EAST-ADL [4], SYSML [16, 9], AML [3]). Some propose connections with the emerging standard AUTOSAR (*http://www.autosar.org*), rely on it for the implementation and build on top some requirement and tracability facilities. Others provide analysis models and tools. However, the ever increasing cost of software in domains like automotive calls for a single framework that would bring together all tools and models and interprete them consistently. Building on the numerous existing, close to maturity, UML (Unified Modeling Language) tools seems like a good way to avoid building and maintaining separate domain-specific graphical editors when graphical aspects should be managed once and for all.

UML is becoming more and more popular for the design and modeling of software systems. Being a general-purpose language, it lacks key features to model software of the real-time and embedded (RTE) domain. The recently adopted UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [15] aims at bringing in the missing features. The goal has never been for MARTE to become the unified (universal) language for modeling RTE systems but rather to be a *pivot* that brings interoperability between the existing languages and formalisms of the RTE domain. The belief is that one single language or model of computation would never be able to cover all aspects for RTE systems whatever good it is for a given specific aspect (schedulability, dependability, requirement ... analyses).

In a continuous effort, we have been trying to show how to use MARTE and its time model [1] to define the semantics of existing models of the RTE domain formally. Following some similar work on AADL [2], this paper focuses on EAST-ADL and explains how to express the semantics of EAST-ADL timing requirements in MARTE. Based on this semantics, our constraint solver can detect inconsistencies and execute the timing requirements when there is no constraint violations. Giving a formal semantics to automotive-related models to make the specifications open for direct analysis resembles other work on AUTOSAR [10] and AADL [12]. However, instead of defining another specific time model, we reuse MARTE time model in a consistent way so that different models and tools can be compared and can ultimately interoperate. Another direct benefit of using MARTE, a UML Profile, is that any UML-compliant tool can be used to edit, store, version and maintain all required models of systems under study, from requirements and system specification to implementation models.

We start with a brief overview of EAST-ADL capabilities in Section 2. Then Section 3 introduces MARTE time model. Follows our contribution in Section 4 where MARTE is used to define the semantics of EAST-ADL timing requirements. Finally, Section 5 discusses the benefits of our proposal and possible modeling improvements. An *Anti-lock Braking System* (ABS) is used as a running example.

## 2. Overview of East-ADL2

EAST-ADL (Electronic Architecture and Software Tools, Architecture Description Language) has been initially developed in the context of the East-EEA European project [14]. To integrate proposals from the emerging standard AUTOSAR and from other requirement formalisms like SYSML, a new release called EAST-ADL2 [4, 13] has been proposed by the ATESST project.

Structural modeling in EAST-ADL covers both analysis and design levels. In this paper the focus is on the analysis level and especially on timing requirements.

### 2.1. Timing Requirements

EAST-ADL requirements extend SYSML requirements and express conditions that must be met by the system. They usually enrich the functional architecture with extra-functional characteristics such as variability and temporal behavior. In this paper, we focus on the three kinds of timing requirements available in EAST-ADL:

1. DelayRequirement that constrains the delay "from" a set of entities "until" another set of entities. It specifies the temporal distance between the execution of the earliest "from" entity and the latest "until" entity;

2. RepetitionRate that defines the inter-arrival time of data on a port or the triggering period of an elementary ADLFunction;

3. Input/outputSynchronization that expresses a timing requirement on the input/output synchronization among the set of ports of an ADLFunction. It should be used to express the maximun temporal skew allowed between input or ouput events or data of an ADLFunction.

Timing requirements specialize the meta-class TimingRestriction, which defines bounds on system timing attributes. The timing restriction can be specified as a *nominal* value, with or without a *jitter*, and can have *lower* and *upper* bounds. The jitter is the maximal positive or negative variation from the nominal value. A bound is a real value associated with an implicit time unit (ms, s, ...).

### 2.2. Example

As an illustration, we consider an Anti-lock Braking System (ABS). This example and the associated timing requirements are taken from the ATESST report on EAST-ADL timing model [6]. The ABS architecture consists of four sensors, four actuators and an indicator of the vehicle speed. The sensors ( $i_{fl}$, $i_{fr}$, $i_{rl}$, $i_{rr}$) measure the rotation speed of the vehicle wheels. The actuators ($o_{fl}$, $o_{fr}$, $o_{rr}$ and $o_{rl}$) indicate the brake pressure to be applied on the wheels. The FunctionalArchitecture is composed of FunctionalDevices for sensors and actuators and an ADLFunctionType for the functional part of the ABS. An ADLOutFlowPort provides the vehicule speed (speed).

The execution of the ABS is triggered by $R$. Parameter $L_s$ measures the latency of sensor sampling. The values of the four sensors involved in the ABS must arrive on the input ADLFlowPorts within delay $J_{ii}$ (InputSynchronization). A similar OuputSynchronization delay $J_{oo}$ is represented on the output interface side. The $L_{io}$ represents the delay from the first event on the input set of the ABS until the last event occurrence on the output set. The sampling interval of the sensor is given by parameter $H$. These parameters are modeled by timing requirements characterized by timing values or intervals with jitters.

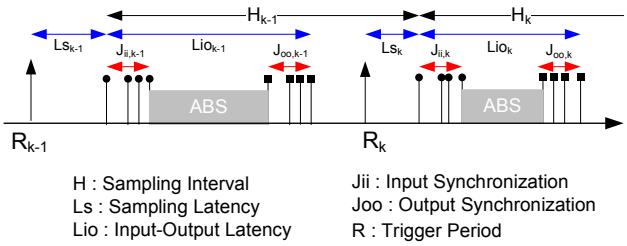


**Figure 1. Timing model of the ABS**

## 3. MARTE Time Model

MARTE defines a broadly expressive *Time Model* that provides for a generic timed interpretation of UML models. In this presentation we focus on *clocks* and *clock constraints*. MARTE time model is described in a previous paper [1]. It defines time structure as a set of interdependent clocks. Interdependency results from relationships existing between instants $i$ and $j$ from different clocks: precedence ($i \prec j$), coincidence ($i \equiv j$) or exclusion ($i \# j$). In practice, such relationships are specified with the dedicated language CCSL. It is briefly presented in the next subsection.

### 3.1. CCSL

CCSL (Clock Constraint Specification Language) is a non normative language annexed to MARTE specification. It is a declarative language that specifies *constraints* imposed on the clocks (activation conditions) of a model. These constraints can be classified into four categories: *synchronous*, *asynchronous*, *mixed*, and *non-functional*.

**Synchronous clock constraints** rely on *coincidence*. *Subclocking* is such a constraint: each instant of the *subclock* must coincide with one instant of the *superclock*. The

mapping must be order-preserving. To model the ABS (see Section 4) we need two other synchronous constraints: discretizedBy and isPeriodicOn. The former discretizes a dense clock. It is mainly used to derive a discrete chronometric clock from $IdealClk$. $IdealClk$ is a dense chronometric clock, predefined in MARTE Time Library, and supposed to follow "physical time" faithfully. For instance:

$$\mathsf{Clock}\ c_{10} = IdealClk\ \mathsf{discretizedBy}\ 0.0001 \qquad (1)$$

Eq. 1 specifies that $c_{10}$ is a discrete chronometric clock whose period is 0.0001 s, where s is the time unit associated with $IdealClk$, therefore $c_{10}$ is a 10 kHz clock.

$$\mathsf{Clock}\ c_1\ \mathsf{isPeriodicOn}\ c_{10}\ \mathsf{period}\ 10 \qquad (2)$$

Eqs. 2 reads that there is a tick of $c_1$ every $10^{th}$ ticks of $c_{10}$ (*i.e.,* $c_1$ is a 1 kHz clock).

**Asynchronous clock constraints** are based on *precedence*, which may appear in a *strict* ($\prec$) or a *non-strict* ($\preccurlyeq$) form.

The clock constraint "$a$ isFasterThan $b$" (symbolically denoted by $a \boxed{\preccurlyeq} b$) specifies that $a$ is (non-strictly) *faster than* $b$, that is for all natural number $k$, the $k^{th}$ instant of $a$ precedes or is coincident with the $k^{th}$ instant of $b$ ($\forall k \in \mathbb{N}, a[k] \preccurlyeq b[k]$). "$b$ isSlowerThan $a$" is equivalent to "$a$ isFasterThan $b$".

*Alternation* is a form of mutual precedence. "$a$ alternatesWith $b$" (symbolically denoted by $a \boxed{\sim} b$) states that $\forall k \in \mathbb{N}^\star, a[k] \prec b[k] \prec a[k+1]$.

**Mixed clock constraints** combine coincidence and precedence. Given 2 clocks $a$ and $b$, "$c = \mathsf{inf}(a,b)$" is the slowest clock faster than both $a$ and $b$ ($\forall k \in \mathbb{N}^\star, c[k] \equiv$ if $a[k] \preccurlyeq b[k]$ then $a[k]$ else $b[k]$). Similarly, "$d = \mathsf{sup}(a,b)$" is the fastest clock among all clocks slower than both $a$ and $b$ $\big(\forall k \in \mathbb{N}, d[k] \equiv$ if $a[k] \preccurlyeq b[k]$ then $b[k]$ else $a[k]\big)$. Most of the time, inf and sup clocks are neither $a$ nor $b$. inf and sup are easily extended to sets of clocks.

Another mixed clock constraint enforces delayed coincidences. "$c = a$ delayedFor $n$ on $b$" imposes $c$ to tick synchronously with the $n^{th}$ tick of $b$ following a tick of $a$. It is considered as a mixed constraint since $a$ and $b$ are not assumed to be synchronous.

**Non Functional Property** constraints apply to chronometric clocks. While $IdealClk$ is supposed to be perfect, an actual clock may have flaws. For instance, its period may not be strictly constant with respect to $IdealClk$. CCSL introduces special constraints to specify *stability*, *drift*, *offset*... of chronometric clocks. Examples of stability are given in Section 4.

**Stochastic parameters** are available in CCSL. Nondetermism introduced by such parameters may reflect a hidden detail or a partial knowledge about the actual constraints. The *uniform* distribution is often used to represent a tolerance interval on a duration.

### 3.2. TimeSquare

TIMESQUARE is the software environment we propose to deal with MARTE time model and CCSL. TIMESQUARE is an Eclipse plugin that has four main functionalities: 1) interactive clock-related specifications, 2) clock constraint checking, 3) generation of a solution, 4) displaying and exploring waveforms.

A wizard, included in TIMESQUARE, facilitates clock definitions, clock constraint specifications, model element browsing, and parameter setting.

The second functionality checks constraint sanity and is called when the above mentioned wizard is not used.

The third functionality relies on a *constraint solver* that yields a satisfying execution trace or issues an error message in case of inconsistency. The traces are given as waveforms written in an IEEE standard textual format [5] for dumpfiles (called VCD). The solver intensively uses Binary Decision Diagrams (BDD) to manipulate boolean equations induced by CCSL clock constraints.

## 4. Modeling East-ADL requirements in CCSL

### 4.1 Clocks and events

The term clock used in CCSL may be misleading and deserves to be further discussed. Every event on which we want to attach time constraints can be associated with a clock. Event is taken here in the very broad sense (as in UML) to denote something that happens (the start of an action, the receipt of a message, ...). In that case, CCSL clocks represent the set of instants at which the concerned event occurs. CCSL clock constraints are relations amongst instants of the related clocks.

Overall, there are two kinds of relations supported by EAST-ADL timing requirements. Either, a timing requirement specifies a temporal relation between two successive occurrences of a single event (*i.e.,* two successive instants of the same clock), or between occurrences of different events. In the latter case, it is mostly between events that occur at the same rate and constraints relate the $i^{th}$ occurrence of one event and the $i^{th}$ occurrence of other events. Additionally, all requirements involve physical time but most also induce logical relations. A part of our contribution is to make explicit these logical relations by applying the adequate CCSL constraints. This section shows how CCSL can express the example described in Section 2.

### 4.2 Repetition rate

A RepetitionRate concerns successive occurrences of the same event (data arriving to or departing from a port, triggering of a function). In all cases, it consists in giving a nominal duration between two successive occurrences/instants of the same event/clock. When the duration is specified in terms of number of occurrences of another event (for instance, number of clock cycles), CCSL relation isPeriodicOn must be used. When the duration is given in seconds (time unit s), then relation discretizedBy must be used. The two relations can also be combined to give both logical and physical constraints. Eq. 2 (Section 3.1) builds a discrete, 1 kHz chronometric clock $c_1$. Eq. 3 uses $c_1$ to specify a 5 ms repetition rate for function $f$, where $f.start$ is a clock associated with the beginning of function $f$. This equation defines $f.start$ as being a subclock of $c_1$ five times less frequent.

$$f.start \text{ isPeriodicOn } c_1 \text{ period } 5 \tag{3}$$

MARTE stereotype TimedProcessing enables the association of a clock with the start event of a given behavior. If a jitter is associated with the period, CCSL relation hasStability should be used: $f.start$ hasStability $0.2$. This stability property combined with Eq. 3 states that the jitter on the nominal 5 ms period is 1 ms, thus expressing the EAST-ADL repetition rate.

### 4.3 Delay requirements

A DelayRequirement constrains the delay between a set of inputs and a set of outputs. Input (resp. output) synchronizations are a specialization without outputs (resp. inputs). At each iteration, all inputs and outputs must occur. So for each iteration $i$, it is a matter of constraining the temporal distance between the $i^{th}$ occurrences of two events (occurring on inputs or outputs). These requirements concern two sets of events and imply a constraint between the earliest occurrence out of the first set (CCSL relation inf) and the latest occurrence out of the second set (CCSL relation sup). In our example, function abs has four inputs, one for each wheel. Eqs. 4–5 specify the *inf* and *sup* for the inputs. Analogous equations exist for the outputs.

$$\text{Clock } i_{inf} = \text{inf}(i_{fl}, i_{fr}, i_{rl}, i_{rr}); \tag{4}$$

$$\text{Clock } i_{sup} = \text{sup}(i_{fl}, i_{fr}, i_{rl}, i_{rr}); \tag{5}$$

CCSL operator delayedFor builds from an initial clock a delayed clock for a given duration. Combining delayedFor with isFasterThan allows for specifying distances between two clock instants. Here again, distance are *a priori* expressed in number of ticks of a reference clock. For instance, Eq. 6 denotes a delay requirement and states that the maximum *end-to-end latency* of the function abs is 3 ms. It characterizes the requirement $L_{io}$.

$$o_{sup} \boxed{\preccurlyeq} (i_{inf} \text{ delayedFor } 3 \text{ on } c_1) \tag{6}$$

When the latency is not a nominal value, similar mechanisms as the one explained in the previous subsection can be exploited to define lower/upper bounds and jitters.

### 4.4 Causal relationships

An ADLFunctionType implies a data flow execution. The flow goes from sensors to actuators and is repeated forever. This causal flow can be modeled in CCSL with the precedence-based relation alternatesWith ($\boxed{\sim}$). For instance, equations below state that each release R must be followed by one and only one capture of each input, one execution of the function $abs$ and one output emission and that all these actions must be performed before the next release. The last equation specifies that all inputs must occur before any output is emitted.

$$R \boxed{\sim} i_{inf} \qquad R \boxed{\sim} i_{sup} \qquad R \boxed{\sim} abs$$
$$R \boxed{\sim} o_{inf} \qquad R \boxed{\sim} o_{sup} \qquad i_{sup} \boxed{\sim} o_{inf}$$

## 5. Discussion and results

The previous section has shown how to express the timing requirements of the ABS system in a *declarative* way using CCSL. This specification can be used as a golden model to validate existing realizations of the system. With TIMESQUARE, we may go a step further towards *executable specifications*. Starting with timing requirements expressed in CCSL, we build a refined model of the timed behavior of the ABS system. Refining a constraint may consist in fixing some parameter values, in accordance with the initial specification. Then, TIMESQUARE simulates the refined model, possibly revealing inconsistency in the specification.

The ABS can be modeled as a periodic *data flow* system, or, because of the *jitters*, an almost periodic one. MARTE clocks and clock constraints can easily capture such a data flow behavior. We use *Timers*, a specialization of the constraint delayedFor, for modeling *latencies* associated with communications and processings. For instance, $i = R$ delayedFor $L_s$ on $c_{10}$ reads that $R$ causes the availability of a sample (clock $i$) with a latency $L_s$. Note that $L_s$ is a stochastic delay defined here by a uniform distribution Uniform$(10..30)$. Since the delay is measured on clock $c_{10}$ (10 kHz), the above constraint states that $1\ ms \leq L_s \leq 3\ ms$, in accordance with the given timing specifications.

Clock $i$ is not part of the ABS specification. We have introduced this auxiliary clock to respect faithfully Figure 1 borrowed from the ATESST example. Since there

are 4 values to acquire, instead of one clock ($i$), we had to consider 4 clocks ($i_{fl}, i_{fr}, i_{rl}, i_{rr}$) each of them being delayed from $i$ for a random duration Uniform$(0..5)$ on $c_{10}$ (*i.e.,* $0 \leq delay \leq 0.5\ ms$). This gives rise to the *input synchronization jitter* $J_{ii}$ forced to be less than $0.5\ ms$.

The current version of TIMESQUARE generates a sequence of steps that satisfies the set of constraints. An inconsistent specification can lead to a *deadlock*: after a finite sequence of steps, the simulation gets at a point where all the clocks are disabled (not allowed to tick). With the values given in the specification, we have encountered such a deadlock caused by the violation of an alternation: $R$ tries to occur while $o_{sup}$ has not occurred yet. $R$ being the (indirect) cause of all other events, the simulation cannot proceed on. This diagnostic is facilitated by the generation of an anomaly report file that contains the history upto the deadlock and the state of all clocks at the faulty step. Raising the period of $R$ from $7\ ms$ to $8\ ms$ allows $o_{sup}$ to occur in time and the simulation runs without abortion. Of course, this is not a proof of correctness. A simulation is conclusive only when it reveals a counter-example. Note that the initial specification allowed only $5\ ms$ to period $R$, which was certainly insufficient and surely rejected by the simulator.

An inconsistent specification may also lead to a "*livelock*": a subset of clocks are mutually exclusive so that they cannot tick any more. This behavior is easily spotted on the simulation display by the persistent absence of pulses for these clocks. This is only a *presumption* of livelock, that should be confirmed or invalidated by inspection of the simulation trace.

## 6. Conclusion

The AUTOSAR community has screened the Time model proposed by the ATESST project and identified a number of delicate issues. Our approach meets these conclusions and can be considered as a proposal to solve some of the issues.

In this paper, we have expressed the semantics of EAST-ADL *timing requirements* using MARTE CCSL. More details about our approach are available in a research report [8]. Making the semantics formal allows for an automatic detection of logical inconsistencies. Then, requirements have been refined into an executable specification to run simulations and explore the dynamic properties of acceptable executions. Executable specifications certainly help detecting errors at early stages. Using a pivot language, like CCSL, allows for better interoperability and builds on the large number of UML-based modeling frameworks.

Even though the simulation can detect some errors in the specification, it should be combined with exhaustive formal analyses. For instance, livelock detection can be automated with model-checking approaches, provided that the state-space is finite. TIMESQUARE already generates the state-space at each step. For now, our default simulation policy consists in randomly selecting a minimal set of fireable clocks at each simulation step. Other policies must be implemented and combined with validation of properties.

## References


[1] C. André, F. Mallet, and R. de Simone. Modeling time(s). In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *MoDELS*, volume 4735 of *Lecture Notes in Computer Science*, pages 559–573. Springer, 2007.

[2] C. André, F. Mallet, and R. de Simone. *Modeling of AADL data-communications with UML Marte*, volume 10 of *LNEE*, chapter 11, pages 150–170. Springer, May 2008.

[3] P. Braun and M. Rappl. A model-based approach for automotive software development. In P. P. Hofmann and A. Schürr, editors, *OMER*, volume 5 of *LNI*, pages 100–105. GI, 2001.

[4] P. Cuenot, D. Chen, S. Gérard, H. Lönn, M.-O. Reiser, D. Servat, C.-J. Sjostedt, R. T. Kolagari, M. Torngren, and M. Weber. Managing complexity of automotive electronics using the East-ADL. In *Proc. of the 12th IEEE Int. Conf. on Engineering Complex Computer Systems (ICECCS'07)*, pages 353–358. IEEE Computer Society, 2007.

[5] IEEE Standards Association. *IEEE Standard for Verilog Hardware Description Language*. Design Automation Standards Committee, 2005. IEEE Std 1364TM-2005.

[6] R. Johansson, H. Lönn, and P. Frey. ATESST timing model. Technical report, ITEA, 2008. Deliverable D2.1.3.

[7] H. Lönn and U. Freund. *Automotive Architecture Description Language*, chapter 9. CRC Press, December 2008.

[8] F. Mallet, C. André, and M.-A. Peraldi-Frati. Marte CCSL and East-ADL2 timing requirements. Research Report 6781, INRIA, December 2008.

[9] OMG. *Systems Modeling Language (SysML) Specification 1.1*. Object Management Group, May 2008. OMG document number: ptc/08-05-17.

[10] K. Richter. Defining a timing model for AUTOSAR - status and challenges. In W. Maalej and B. Brügge, editors, *Software Engineering (Workshops)*, volume 122 of *LNI*, pages 93–97. GI, 2008.

[11] SAE. *Architecture Analysis and Design Language*. Society of Automotive Engineers, June 2006. Document AS5506/1.

[12] O. Sokolsky, I. Lee, and D. Clarke. Schedulability analysis of AADL models. In *IPDPS*. IEEE, 2006.

[13] The ATESST Consortium. East-ADL2 specification. Technical report, ITEA, March 2008. http://www.atesst.org, 2008-03-20.

[14] The East-EEA Project. Definition of language for automotive embedded electronic architecture approach. Technical report, ITEA, 2004. Deliverable D.3.6.

[15] The ProMARTE Consortium. *UML Profile for MARTE, beta 2*. Object Management Group, June 2008. OMG document number: ptc/08-06-08.

[16] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. The MK/OMG Press, Burlington, MA, USA., 2008.